**1 . what is js?**

Ans: [JavaScript](https://www.geeksforgeeks.org/javascript/) is a versatile, [dynamically typed](https://www.geeksforgeeks.org/what-is-a-typed-language/)programming language used for interactive web applications, supporting both client-side and server-side development, and integrating seamlessly with [HTML](https://www.geeksforgeeks.org/html-tutorial/),[CSS,](https://www.geeksforgeeks.org/css-tutorial/) and a rich standard library.

* JavaScript is a [single-threaded](https://www.geeksforgeeks.org/why-javascript-is-a-single-thread-language-that-can-be-non-blocking/)language that executes one task at a time.
* It is an [Interpreted language](https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/) which means it executes the code line by line.
* The data type of the variable is decided at run-time in JavaScript that’s why it is called dynamically typed.
* (everything in js happens inside an Execution context )
* ( js is a synchronous  [single-threaded](https://www.geeksforgeeks.org/why-javascript-is-a-single-thread-language-that-can-be-non-blocking/)language)

**2. what is ECMA script?**

Ans: ECMAScript (ECMA) is the official standard that defines JavaScript's syntax and features. It ensures consistency across JavaScript engines, with versions like ES6 introducing major improvements.

3. what is ECMAscript and ES6

Ans : No, ECMAScript is the official standard for JavaScript, while ES6 (ECMAScript 2015) is a specific version of that standard. ES6 introduced major updates, but ECMAScript continues to evolve with newer versions like ES7, ES8, and beyond.

**4. what is ES?**

Ans : ES13 (ECMAScript 2022) is the 13th edition of ECMAScript, introducing features like:

* **at() method** for indexing arrays and strings (array.at(-1) for the last element).
* **Top-level await** for using await in modules without wrapping in async functions.
* **Class field declarations** for defining properties directly in class bodies.
* **Object.hasOwn()** as a safer alternative to Object.prototype.hasOwnProperty().

It enhances JavaScript's usability and performance in modern applications.

**5. what is js engine?**

Ans: A **JavaScript engine** is a program that executes JavaScript code. It takes JavaScript source code, compiles or interprets it, and runs it efficiently. Popular JavaScript engines include:

* **V8** (used in Chrome and Node.js)
* **SpiderMonkey** (used in Firefox)
* **JavaScriptCore (Nitro)** (used in Safari)

These engines optimize JavaScript execution for better speed and performance in browsers and environments like Node.js.

**6. what is synchronous?**

Ans: JavaScript is **synchronous** by default, meaning it executes code **line by line**, blocking the next task until the current one finishes. However, it supports **asynchronous** operations using callbacks, promises, and async/await to handle tasks like API calls and timers without blocking execution.

**7. what is execution Context?**

Ans: The **execution context** in JavaScript is an environment where code is executed and evaluated. It includes the **Global Execution Context (GEC)** for the main script and **Function Execution Contexts (FEC)** for each function call, managing variables, scope, and the execution order.

(There are 2 components in EC

First one is memory component and second one is code component

This EC is created in two phases…..1st one is memory creation phase and 2nd one is the code execution phase

**8. what is Call stack and Execution Context stack?**

Ans : The **Execution Context Stack** (also called the **Call Stack**) is a stack data structure that manages the execution of JavaScript code. It stores execution contexts in a **Last In, First Out (LIFO)** order, handling function calls by pushing new execution contexts and removing them when functions return.

**Variables**

Let & const were introduced in 2015 as new features of JavaScript. Before that, var was the only way to declare a variable.

**9. what is variable in js?**

Ans: variable is just a named memory location.

**Var**: The **var** keyword was the traditional way of declaring variables in JavaScript. Variables declared with *var* are function-scoped or globally-scoped, but not block-scoped. This means that variables declared with *var* are hoisted to the top of their function or global scope. This hoisting behavior can sometimes lead to unexpected results and make debugging challenging.

var is function scoped

When we declare a variable using **var** inside a function, we canot access the variable outside the function.

Reassignment and Redefinition using var

In JavaScript, the **var** keyword allows variables to be both reassigned and redefined within the same scope, unlike *let* which only allows reassignment. This behavior of *var* can sometimes lead to unexpected results and bugs in code if not used carefully.

**Let**:

In JavaScript, the **let** keyword is used to declare variables that are block-scoped, meaning they are only accessible within the block in which they are defined. Block scope refers to any code block delimited by curly braces *{}* such as loops, conditionals, or function bodies.

Reassignment and Redefinition using let

Variables declared with **let** can be reassigned, just like those declared with *var*, but they cannot be redeclared within the same block scope. This helps prevent accidental redeclaration of variables and can aid in writing more predictable and maintainable code.

**const**

In JavaScript, the **const** keyword is used to declare constants. Constants are variables whose values cannot be reassigned once they are initialized. This means that once a value is assigned to a constant using *const*, it cannot be changed or reassigned throughout the execution of the script.

Constants declared with *const* must be initialized with a value. Unlike variables declared with *var* or *let*, which can be declared without immediately assigning a value, const requires initialization at the time of declaration.

**10. DataTypes ?**

Ans: JavaScript has **two categories** of data types:

**1. Primitive Data Types (Immutable)**

* **String** → "Hello"
* **Number** → 42, 3.14
* **Boolean** → true, false
* **Undefined** → A variable declared but not assigned a value (let x;)
* **Null** → An intentional empty value (let y = null;)
* **BigInt** → For very large numbers (123n)
* **Symbol** → Unique and immutable values (Symbol('id'))

**2. Non-Primitive (Reference) Data Types**

* **Object** → { name: "John", age: 25 }
* **Array** → [1, 2, 3, 4]
* **Function** → function greet() { return "Hello"; }

Objects, arrays, and functions are mutable and stored by reference.

**11. Operators in JavaScript**

Ans : JavaScript is a dynamic programming language, has various operators to perform operations on data and manipulate values. In this article, we'll learn the diverse types of operators in JavaScript, exploring their functionalities and providing practical examples to enhance your understanding.

Types of operators

Understanding JavaScript operators is the most basic thing for writing efficient and expressive code. By mastering these operators, we'll gain the tools needed to manipulate data, make decisions, and create more dynamic and interactive applications.

🌟 Arithmetic Operators

📍Assignment Operators

🌟Comparison Operators

📍Logical Operators

🌟Bitwise Operators

📍Ternary Operators

**12. Types of literals in JavaScript**

Ans : JavaScript literals serve as syntactic representations for various types of data like objects, string, boolean, array, and more.

But when we talk about the most important JavaScript literals we have the following list:

* String literals

let myName = "Mayank Pandey!";

let message = 'JavaScript is a scripting language.';

Concatenation of strings

let firstName = "Mayank";

let lastName = "Pandey";

let fullName = firstName + " " + lastName;

console.log(fullName);

//Output : Mayank Pandey

>Template literals

let number1 = 10;

let number2 = 20;

let sum = number1 + number2;

console.log(`The sum of ${number1} + ${number2} is equal to ${sum}`);

//Output : The sum of 10 + 20 is equal to 30

..

Print multiline string

```

let string1 = "Mai pal do pal ka shayar hoon";

let string2 = "Pal do pal meri kahani hai";

let multiLineString = `

${string1},

${string2}`

console.log(multiLineString);

Output :

Mai pal do pal ka shayar hoon,

Pal do pal meri kahani hai

* Object literals

let person = {

name: 'Mayank',

age: 21,

isRunning : true

};

* Array literals

let colors = ['orange', 'red', 'aqua', 'green', 'blue'];

let container = [2, "laptop", true, "papers", "watch"];

**13. Loops**

We use loops to perform repeated actions. For example - If you are designed a task printing will be numbers from 1 100, it very hectic to do it manually. Loops help us automate such tasks

for loop ←

loop a block of code number of times

for in loop →

loops through the keys of an object

for of loop

loops through the values of an object

While loop ←

loops a block based on a specific condition

do-while loop

while loop variant which runs at least once.

**14 . Functions in js?**

Ans : a javascript function is a block of code and designed to perform a particular task.

Syntax of function looks like this;

Function myname() {

// code  
}

Function myname(parameter1 , parameter2) {

// code  
}

Myname(“shaikh”,”Firoz”);…………….invocation

Function invocation is a way to use the code inside the function

A function can also return a value. the value is return back to the called

Const sum =(a,b) =>{

Let c=a+b;

Return c;

}

let y= sum(1+2);

console.log(y); ///3

**15. what is DOM?**

Ans: **The HTML DOM (Document Object Model)** is a programming interface that represents the structure of a web page in a way that programming languages like JavaScript can understand and manipulate.

**What Does the HTML DOM Look Like?**

**Imagine your webpage as a tree**

* The document is the root.
* HTML tags like <html>, <head>, and <body> are branches.
* Attributes, text, and other elements are the leaves.

**Why is DOM Required?**

**The DOM is essential because**

* **Dynamic Content Updates:** Without reloading the page, the DOM allows content updates (e.g., form validation, AJAX responses).
* **User Interaction:** It makes your webpage interactive (e.g., responding to button clicks, form submissions).
* **Flexibility:**Developers can add, modify, or remove elements and styles in real-time.
* **Cross-Platform Compatibility:** It provides a standard way for scripts to interact with web documents, ensuring browser compatibility